Електротехнички факултет у Београду
Катедра за рачунарску технику и информатику

Заштита података

- Пројектни задатак 2020/2021. -

Студенти:

Теслић Стефан 2017/0124
Самарџија Сања 2017/0372

Увод

Имплементиран је ПГП алгоритам за слање и пријем порука. У склопу њега омогућене су следеће фунционалности:

- Генерисање новог и брисање постојећег пара кључева
- Увоз и извоз јавног или приватног кључа у .asc формату
- Приказ прстена јавних и приватних кључева са свим потребним информацијама
- Слање поруке (уз обезбеђивање енкрипције и потписивања)
- Примање поруке (уз обезбеђивање декрипције и верификације)

При генерисању новог пара кључева од корисника се поред уноса имена, мејла тражи и одабир величине кључа за RSA алгоритам (**1024, 2048** или **4096** бита). Потом и унос лозинке под којом ће се приватни кључ чувати.

За потписивање се користи **SHA-1** алгоритам који хешира поруку и **RSA** приватни кључ који енкриптује хеш. За верификацију се користи одговарајући јавни кључ да декрипутје хеш, након чега се хеш проверава на једнакост и потврђује потпис.

За енкрипцију поруке се користе алгоритми за симетричне сесијске кључеве. Корисник бира **3DES** или **IDEA** и користи одабрани кључ који се потом енкриптује са примаоцевим јавним кључем. Асиметрични кључеви за енкрипцију су у нашем случају подкључеви добијени **RSA** алгоритмом.

Главни пакет **etf.openpgp.ts170124dss170372d:**
Пакет **pgp:**
PGP – класа која садржи функције за потпис, енкрипцију, декрипцију и верификацију фајла

```
/**
 * Encrypt and/or sign the file based on preferences
 *
 * @param sign  if true sign file using {@code signKeyID}, otherwise no effect
 * @param encrypt  if true encrypt file using {@code data}, otherwise no effect
 * @param radix64  if true encrypted data will be encoded with {@link
ArmoredOutputStream}
 * @param compress  if true data will be compressed before encryption using
 *                  {@code ZIP} algorithm {@link CompressionAlgorithmTags}
 * @param algorithm  algorithm to be used for encryption {@link
SymmetricKeyAlgorithmTags}
 * @param data  {@link EncryptionWrapper} data of public keys for encryption
 * @param fileLocation  path to the {@link File} you wish to encrypt and/or sign
 * @param signKeyID  ID of key used to sign the given file
 * @param passphrase  password used to extract {@link PGPPrivateKey} for signature
 * @throws PGPException
 * @throws IOException
 * @throws IllegalArgumentException
 */
public static void signatureAndEncryption(boolean sign,
                                          boolean encrypt,
                                          boolean radix64,
                                          boolean compress,
                                          int algorithm,
                                          ArrayList<EncryptionWrapper> data,
                                          String fileLocation,
                                          long signKeyID,
                                          String passphrase) throws PGPException,
IOException, IllegalArgumentException
```

Функција за потпис и енкрипцију фајла са изабраним параметрима

```
/**
 *  Decrypt file with given name and verify its signatures
 *
 * @param inputFileName {@code String} for the file to be decrypted
 * @param passphrase {@code String} used to decode the {@link PGPSecretKey}
 * @param fileName {@code String} used to make a new decoded {@link File}
 *                       if file name not present use one from encoded data
 * @return {@link DecryptionVerificationWrapper}
 * @throws IOException
 * @throws PGPException
 * @throws SignatureException
 */
public static DecryptionVerificationWrapper decryptionAndVerification(String
inputFileName,
                                                       String passphrase,
                                                       String fileName) throws PGPException,
IOException
```

Функција за декрипцију и верификацију фајла

```
/**
 * Encrypt the file based on preferences
 *
 * @param fileToEncrypt  path to the {@link File} you wish to encrypt
 * @param publicKeys  array of {@link PGPPublicKey} which you wish to encrypt the data
with
 * @param algorithm  algorithm to be used for encryption {@link
SymmetricKeyAlgorithmTags}
 * @param compress  if true data will be compressed before encryption using
 *                  {@code ZIP} algorithm {@link CompressionAlgorithmTags}
 * @param radix64  if true encrypted data will be encoded with {@link
ArmoredOutputStream}
 * @throws IOException
 * @throws PGPException
 */
private static String encryptFile(String fileToEncrypt,
                        PGPPublicKey[] publicKeys,
                        int algorithm,
                        boolean compress,
                        boolean radix64) throws IOException, PGPException
```

Функција за енкрипцију фајла са изабраним параметрима

```
/**
 * Sign file with provided private key
 *
 * @param fileToSign name of the signed {@link File} to sign
 * @param privateKey {@link PGPPrivateKey} used to sign the file
 * @param publicKey {@link PGPPublicKey} used to sign the file
 * @param radix64 if true encrypted data will be encoded with {@link ArmoredOutputStream}
 * @param compress  if true data will be compressed before encryption using
 *                        {@code ZIP} algorithm {@link CompressionAlgorithmTags}
 * @return {@code String} name of signed {@link File}
 * @throws IOException
 * @throws PGPException
 */
private static String signFile(
        String fileToSign,
        PGPPrivateKey privateKey,
        PGPPublicKey publicKey,
        boolean radix64,
        boolean compress) throws IOException,PGPException
```

Функција за потпис фајла са изабраним параметрима

```
/**
 * Sign file with provided private key and encrypt the file based on preferences
 *
 * @param fileToSign name of the signed {@link File} to sign
 * @param privateKey {@link PGPPrivateKey} used to sign the file
 * @param publicKey {@link PGPPublicKey} used to sign the file
 * @param publicKeys  array of {@link PGPPublicKey} which you wish to encrypt the data
with
 * @param algorithm  algorithm to be used for encryption {@link
SymmetricKeyAlgorithmTags}
 * @param radix64 if true encrypted data will be encoded with {@link ArmoredOutputStream}
 * @param compress  if true data will be compressed before encryption using
 *                      {@code ZIP} algorithm {@link CompressionAlgorithmTags}
 * @return {@code String} name of signed and encrypted {@link File}
 * @throws IOException
 * @throws PGPException
 */
private static String signAndEncrypt(String fileToSign,
                                     PGPPrivateKey privateKey,
                                     PGPPublicKey publicKey,
                                     PGPPublicKey[] publicKeys,
                                     int algorithm,
                                     boolean radix64,
                                     boolean compress) throws IOException,PGPException
```

Функција за потпис и енкрипцију фајла са изабраним параметрима

```
/**
 *  Decrypt file with given name and verify its signatures
 *
 * @param inputFileName {@code String} for the file to be decrypted
 * @param secretKeyFileName {@code String} for the secret key to be found
 * @param publicKeyFileName {@code String} for the public key to be found
 * @param passphrase {@code String} used to decode the {@link PGPSecretKey}
 * @param outputFileName {@code String} used to make a new decoded {@link File}
 *                       if file name not present use one from encoded data
 * @return {@link DecryptionVerificationWrapper} signature, signature verification and
integrity check data
 * @throws IOException
 * @throws PGPException
 * @throws SignatureException
 */
private static DecryptionVerificationWrapper decryptAndVerify(String inputFileName,
                                      String secretKeyFileName,
                                      String publicKeyFileName,
                                      String passphrase,
                                      String outputFileName) throws IOException,
PGPException
```

Функција за декрипцију и верификацију потписа (ако постоји) фајла

Пакет **utility:**

PGPutil - класа која садржи функције за компресију и проналажење приватног кључа

```
/**
 * Compresses file to {@link PGPLiteralData}, writes it to a file
 * and returns it as a byte array
 *
 * @param fileName  {@code String} name of file to which compressed
 *                          {@link PGPLiteralData}  data is written
 * @return {@code byte[]} compressed file as byte array
 * @throws IOException
 */
public static byte[] compressFile(String fileName) throws IOException
```

Функција која компресује податке и претвара их у *PGPLiteralData*

```
/**
 * Finds a secret key for keyID in {@link PGPSecretKeyRingCollection} and decrypts it.
 * If such key exists the function returns a decrypted private key {@link PGPPrivateKey}
 * null otherwise
 *
 * @param secretKeyRingCollection  {@link PGPSecretKeyRingCollection} to search in
 * @param keyID {@code long}  keyID of the key we are looking for
 * @param passphrase {@code String}  passphrase for secret key decryption
 * @return {@link PGPPrivateKey} or {@code null}
 *          the private key if found or null if no such key found
 * @throws PGPException
 */
public static PGPPrivateKey findPrivateKey(PGPSecretKeyRingCollection
secretKeyRingCollection,
                                           long keyID, String passphrase) throws
PGPException
```

Функција која проналази приватан кључ на основу тајног

User – помоћна класа за податке о кориснику

RSA – класа за генерисање парова кључева, имплементирана као синглтон

```
    /**
     * Postavlja novi keysize.
     * Default je 1024
     * @param keySize Velicina kljuca, uzima se enum KeySizes koji je static
     * @return RSA - graditelj pattern
     */
    public RSA RSA_SetKeySize(KeySizes keySize);
 /**
     * Ovo je zapravo ono e sto stavljamo
     * Default je "10001"
     * @param exponent U string formatu, konvertuje se u hex
     * @return Builder pattern, vraca singleton
     */
    public RSA RSA_SetPublicExponent(String exponent);

    /**
     * Generisanje parova kljuceva.
     * Za promenu velicine kljuca, prethodno je neophodno da se
     * @return
     * @throws NoSuchAlgorithmException
     */
    public KeyPair RSA_KeyGenerator() throws NoSuchAlgorithmException;

    /**
     * Ova metoda genrise par kljuceva uz pomoc RSA.
     * Generise se PGPKeyPair koji u sebi sadrzi dosta nekih info kao na primer,
     * koji alg je koriscen za generisanje, javni i privatni, key id itd..
     *
     * @return PGPKeyPair
     * @throws Exception
     */
    public PGPKeyPair RSA_PGPKeyGenerator() throws Exception;
```

Пакет **utility.helper:**

DecryptionVerificationWrapper – класа која служи за враћање података о успешности декрипције односно верификације

EncryptionWrapper – класа која служи за приказ кључева у графичком интерфејсу

PasswordDialog extends Dialog<String> – класа која је преузета са интернета за приказ дијалога за унос лозинке https://gist.github.com/drguildo/ba2834bf52d624113041

Пакет **utility.KeyManager:**

ExportedKeyData - класа која прилагођава податке ПГП-а за лакше коришћење даље у програму.

Keyring - интерфејс који има декларације свих метода које је неопходно имплементирати. У овој секцији ће бити описано шта која метода ради. Неће се понављати за саму имплементацију интерфејса

```
/**
     * Dodavanje tajnih kljuceva. Zamisljeno je da se radim preko
     * secret KEYRING-a. Po standardu - keyring se sastoji od Master kljuca + subkljucevi.
     * Ja sam napravio da radimo SAMO sa master kljucem. Zasto -
pricali smo samo o jednim kljucem na predavanjima
     * i vezbama.
     * <p>
     * Dalje, prsledjuje se secret keyring i unutar metode mi pravimo novu listu u koju prvo
     * prekopiramo sve sto se nalazi u KOLEKCIJI keyring-ova i dodamo u listu novi kljuc.
     * Potom samo instanciramo novi SecretKeyCollection
     *
     * @param secretKey
     * @throws IOException
     * @throws PGPException
     */
    void addSecretKey(PGPSecretKeyRing secretKey) throws IOException, PGPException;

    /**
     * Pogledaj sta pise za dodavanje tajnih kljuceva, isto je sve, samo sto se u ovom slucaj
u
     * radi sa PublicKeyRing i publick keyring kolekcijom. Isti je "algoritam";
     *
     * @param publicKey
     * @throws IOException
     * @throws PGPException
     */
    void addPublicKey(PGPPublicKeyRing publicKey) throws IOException, PGPException;

    /**
     * U ovoj metodi treba uraditi proveru da li je kljuc sa dostavljenim KeyId postoji
     * ili ne! TODO
     * Pozivom ove metode prvo trazimo secret key i onda ga prosledjujemo istoimenoj metodi
     *
     * @param KeyId
```

```
     * @param password
     * @throws PGPException
     * @throws IncorrectKeyException
     */
    void removeSecretKey(long KeyId, String password) throws PGPException, IncorrectKeyExcept
ion, KeyNotFoundException;

    /**
     * Sacuva po defaultu keyringove u default fajlove
     * @throws IOException
     */
    void saveKeys() throws IOException;

    /**
     * Ova metoda prvo radi ekstrakciju privatnog kljuca.
     * Razlika izmedju tajnog i privatnog kljuca je u tome sto je tajni kljuc sifrovan privat
ni kljuc.
     * Dakle, ekstrakcijom privatnog kljuca treba da dekriptujemo tajni kljuc. Tu onda prover
avamo
     * da li je dobar password. Ako password nije dobar, baca se IncorrectKeyException!
     * <p>
     * Ako je kljuc dobar, onda ulazimo u narednu metodu koja se zove removeGivenSecretKeyFro
mCollection.
     * Za vise detalja o ovoj metodi, pogledati sam KeyringManager.java kako je ova metoda pr
ivatna.
     *
     * @param keyRing
     * @param password
     * @throws IncorrectKeyException
     */
    void removeSecretKey(PGPSecretKeyRing keyRing, String password) throws IncorrectKeyExcept
ion;

    /**
     * U ovoj metodi treba uraditi proveru da li je kljuc sa dostavljenim KeyId postoji
     * ili ne! TODO
     * Pozivom ove metode prvo trazimo public key i onda ga prosledjujemo istoimenoj metodi
     *
     * @param KeyId
     * @throws PGPException
     * @throws IOException
     */
    void removePublicKey(long KeyId) throws PGPException, IOException;

    /**
     * TODO Tu treba provera za postojanje kljuca.
     * Ako kljuc postoji trebalo bi da se pozove removeGivenPublicKeyFromCollection. pogledat
i .java fajl.
     * Ovo je privatna metoda;
     *
     * @param keyRing
```

```
     * @throws IOException
     * @throws PGPException
     */
    void removePublicKey(PGPPublicKeyRing keyRing) throws IOException, PGPException;

    // Key generation

    /**
     * Treba dostaviti PGP pair od RSA utility klase. Ima primer kako se generise par kljucev
a u testovima,
     * to je sve sto treba da se zna, ne znam da li sam napisao komentae tamo.
     * <p>
     * Ova metoda prvo pravi flegove kao sto su za sta kljuc sluzi, koji su simetricni algori
tmi koje perferiramo,
     * hash alg, postavlja se datum isteka kljuca - default-
no 1 godina. To je hash flags, onda posle toga
     * idu non hash flagovi koji se vezuju za subkeys (koje ne koristimo tkd su tehnicki suvi
sni)
     * <p>
     * Posle toga se pravi Keyring generator koji ima zadatak da sve to upakuje u keyring-
ove.
     * <p>
     * Svi podaci mogu da se izvuku iz PGPSecretKeyring jer secret key sadrzi i javni kljuc k
oji sadrzi
     * sve info o javnom kljuc.
     * <p>
     * Posle iz generatora generisemo keyringove (sa svim tim upakovanim flagovima/info)
     * <p>
     * Kada generisemo - dodajemo keyringove u public key i secret key kolekcije.
     * <p>
     * Kada se to uradi, poziva se saveKeys koji samo pravi dva fajla i cuva u projektu
     * <p>
     * TODO Ekstrahovati nazive fajlova kao final staitc
     * @param masterKey
     * @param subKey
     * @param username
     * @param email
     * @param password
     * @throws PGPException
     * @throws IOException
     * @return
     */
    ExportedKeyData makeKeyPairs(PGPKeyPair masterKey, PGPKeyPair subKey, String username, St
ring email, String password) throws PGPException, IOException;

    // Helper methods

    /**
     * Kao sto ime kaze, helper funkcija koja cuva kljuceve na preodredjeno mesto sa nepredod
redjenim nazivima fajla.
```

```
     * Ova metoda konkretno poziva istoimenu metodu kako se ova metoda poziva kada zelimo da
sacuvamo javni i tajni
     * keyting collection sto se interno nalazi u KeyringManager-u.
     * <p>
     * TODO: Provera da li su null
     *
     * @param publicKeyFileLocation
     * @param secretKeyFileLocation
     * @throws IOException
     */
    void saveKeys(String publicKeyFileLocation, String secretKeyFileLocation) throws IOExcept
ion;

    /**
     * U ovoj metodi se konkretno samo prave ourput stream-ovi, file streamovi.
     * Potom samo pisanje fajla delegiramo writeKeyToFIle funkciji koja obavlja sam zadatak p
isanja.
     * Metoda je privatna tkd pogledaj .java fajl za vise info
     *
     * @param publicKeyRings
     * @param secretKeyRings
     * @param publicKeyFileLocation
     * @param secretKeyFileLocation
     * @throws IOException
     */
    void saveKeys(PGPPublicKeyRingCollection publicKeyRings, PGPSecretKeyRingCollection secre
tKeyRings, String publicKeyFileLocation, String secretKeyFileLocation) throws IOException;

    /**
     * Ovu metodu treba koristiti iskljucivo kada zelimo da ispisemo celu kolekciju koja se n
alazi u nasem
     * menadzeru,
     * TODO: Provera null za kolekcije
     *
     * @return
     */
    ArrayList<ExportedKeyData> generatePublicKeyList();

    /**
     * Ova metoda ima zadatak da formatira kolekcije za jednostavnije baratanje kljucevima.
     * Vraca listu svih kljuceva na malo cudan nacin.
     * <p>
     * Naime, pretpostavka je da se svaki tajni kljuc (pod ovo mislim keyID tajnog kljuca) na
lazi u
     * public key kolekciji, ali javni kljuc ne mora da se nalazi u private key kolekciji.
     * <p>
     * U svakom slucaju -> prolazimo kroz sve javne kljuceve, ubacujemo u novi niz tako sto
     * uzimamo javni kljuc i prosledjujemo ga metodi extractDataFromKey (pogledaj .java za vi
se info)
     * i on nam formatira informacije tako sto napravi objekat ExportedKeyData koji ima neke
atribute od znacaja
```

```
     * za ispis.
     * <p>
     * Potom, prolazimo kroz sve tajne kljuceve, ako naidjemo na keyID koji se nalazi u listi
, obavezno
     * stavljamo flag da je master, iliti (mozes da shvatis zbog ovog "frameworka") za taj Ke
yID posedujemo i privatni
     * kljuc sto znaci da mozemo da potpisujemo sa njim.
     *
     * @param publicKeyRings
     * @param secretKeyRings
     * @return
     */
    ArrayList<ExportedKeyData> generatePublicKeyList(PGPPublicKeyRingCollection publicKeyRing
s, PGPSecretKeyRingCollection secretKeyRings);

    // ------------------------
    // --- Sledece tri metode --
    // --- Nisu testirane ------
    // ------------------------

    /**
     * Obavezno treba proveriti van ove metode da li je pronadjen kljuc ili ne
     * Nista
     *
     * @param keyId
     * @return
     * @throws PGPException
     */
    PGPSecretKey getSecretKeyById(long keyId) throws PGPException;

    /**
     * Obavezno van metode proveriti da li je uspesno dekriptovano -
 dobija se null ako je neuspesno!
     * Dekripcija je efektivno ista kao kod brisanja
     *
     * @param secretKey
     * @param password
     * @return
     */
    PGPPrivateKey decryptSecretKey(PGPSecretKey secretKey, String password);

    /**
     * Isto kao decryptSecret key, samo sto nas ne zanima private key nego
     * samo da li se passwordi poklapaju
     *
     * @param secretKey
     * @param password
     * @return
     */
    boolean checkPasswordMatch(PGPSecretKey secretKey, String password);
```

```
    // Imports and exports

    /**
     * Ova metoda je krindz hehehhehehe
     * <p>
     * Drzi na umu ono -
 nemamo master key i sub kljuceve vec sve radimo sa jednim kljucem te keyring postaje nas klj
uc!
     * <p>
     * TODO: Da li kljuc postoji?
     * <p>
     * Prvo dohvatamo public key iz kolekcije (pretpostavka je da SVI keyID tamo postoje)
     * <p>
     * Onda dolazi onaj krindz deo -
 Pravimo KeyRing tako sto pravimo arraylist sa samo jednim javnim
     * kljucem i ubacujemo u konstruktor keyring-
a i to prosledjujemo nadalje u istoimenu metodu
     *
     * @param KeyId
     * @param os
     * @throws PGPException
     * @throws IOException
     */
    void exportPublicKey(long KeyId, OutputStream os) throws PGPException, IOException, KeyNo
tFoundException;

    /**
     * TODO Isto fale neke provere sig
     * <p>
     * Pozivamo write Key to file metodu, naci u .java
     *
     * @param pgpPublicKey
     * @param os
     * @throws PGPException
     * @throws IOException
     */
    void exportPublicKey(PGPPublicKeyRing pgpPublicKey, OutputStream os) throws PGPException,
 IOException;

    /**
     * Sejm shit kao i za public
     *
     * @param KeyID
     * @param outputStream
     * @throws PGPException
     * @throws IOException
     * @throws KeyNotFoundException
     */
    void exportSecretKey(long KeyID, OutputStream outputStream) throws PGPException, IOExcept
ion, KeyNotFoundException;
```

```
    /**
     * Sejm shit kao i za public
     *
     * @param key
     * @param outputStream
     * @throws PGPException
     * @throws IOException
     * @throws KeyNotFoundException
     */
    void exportSecretKey(PGPSecretKeyRing key, OutputStream outputStream) throws IOException;

    /**
     * Ovu metodu ne treba koristiti, konsultovati importSecretKeyring metodu
     *
     * @param secretKey
     * @throws IOException
     * @throws PGPException
     * @Depricated
     */
    void addSecretKey(InputStream secretKey) throws IOException, PGPException;

    /**
     * Ovu metodu ne treba koristiti, konsultovati importPublicKeyring metodu
     *
     * @param publicKey
     * @throws IOException
     * @throws PGPException
     * @Depricated
     */
    void addPublicKey(InputStream publicKey) throws IOException, PGPException;

    ExportedKeyData importSecretKeyring(InputStream is) throws IOException, PGPException;

    ExportedKeyData importPublicKeyring(InputStream is) throws IOException, PGPException;
```

KeyringManager - класа која производи објекат који води администрацију кључева. Представља имплементацију интерфејса Keyring. Има неколико метода које су специфичне за ову класу налазе се у наставку

```
    /**
     * ArmoredOutputStream koristim za sve keyexporte - meni lakse, svima lakse
     * Armored output samo znaci Base64 konverzija.
     *
     * @param outputStream
     * @param encoded
     * @throws IOException
     */
    private static void writeKeyToFile(OutputStream outputStream, byte[] encoded) throws IOEx
ception;


    /**
```

```
     * 1. Napravi novi obj ExportedKeyData
     * 2. Dohvati javni kljuc (radi za priv i javni jer prosledjujemo keyRING)
     * 3. Ekstrahuj user ID (Oblik je Username <email>)
     * 4. Delimo username i email iz user ID
     * 5. Punimo podatke exported data key
     * <p>
     * Vazna napomena, podrazumevano je master Key false,
     * ako se ocekuje master key, treba nekako da se zameni!
     *
     * @param pgpRing
     * @return
     */
    public static ExportedKeyData extractDataFromKey(PGPKeyRing pgpRing);


    /**
     * Ovo koristimo da nadovezemo sekunde vazenja
     * na datum pravljenja kljuca, pogledaj javinu dok za ovo
     *
     * @param date
     * @param seconds
     * @return
     */
    private static Date addSeconds(Date date, Long seconds);

   /**
     * Uzimamo sve kljuceve iz kolekcije, pravimo listu tako sto dodajemo
     * ako nije keyring koji brisemo
     *
     * @param keyRing
     * @throws IOException
     * @throws PGPException
     */
    private void removeGivenSecretKeyFromCollection(PGPSecretKeyRing keyRing) throws IOExcept
ion, PGPException;

/**
     * Isto kao i za private brisanje
     *
     * @param keyRing
     * @throws IOException
     * @throws PGPException
     */
    private void removeGivenPublicKeyFromCollection(PGPPublicKeyRing keyRing) throws IOExcept
ion, PGPException;


/**
     * @param inputStream
     * @throws IOException
     * @throws PGPException
```

```
     * @see <a href="https://stackoverflow.com/questions/28444819/getting-bouncycastle-to-
decrypt-a-gpg-encrypted-message">
     * Odavde je preuzet kod za dekriptovanje
     * </a>
     * @return
     */
    @Override
    public ExportedKeyData importPublicKeyring(InputStream inputStream) throws IOException, P
GPException;
/**
     * Ovo koristim da pretrazimo fajlsistem za trazeni fajl i da vratimo
     * putanju od korenog dir.
     * Na primer ako je root ./
     * U root imamo fajl asdf.txt i folder a i unutar njega a/fdsa.txt
     * ako trazimo asdf.txt on vrasca "asdf.txt"
     * <p>
     * Ako trazimo fdsa.txt, vraca a/fdsa.txt
     *
     * @param filename
     * @return
     */
    private String scanForFile(String filename);
```

Пакет **ExceptionPackage:**

IncorrectKeyException extends Exception – класа која представља изузетак за лош кључ

KeyNotFoundException extends Exception – класа која представља изузетак за кључ који се не може пронаћи

NullObjectException extends Exception – класа која представља изузетак за null објекат

Пакет за тестирање:

PGP_test – класа за тестирање функција класе PGP

RSA_test – класа за тестирање функција класе RSA